Software Verification on the ASIP CAD Example

or
How to Trust Your Team and Yourself?

Ph.D. Aleksandr Penskoi, ITMO University, Russia, 29.05.2021

About myself

- Graduate at ITMO University
  - 2016 — Ph.D. — Research and Development Architectural Style for Design Multi-Level Embedded Systems
- Associate Professor
  - 2017 — At Software Engineering and Computer Systems Faculty, ITMO University
  - 2020 — At ITMO University and Hangzhou Dianzi University Joint Institute
- 2010 — 2017 — Software Engineer at LMT Ltd. (Embedded systems design center)
- 2017 — 2020 — Architect & Senior Developer at National Center for Cognitive Research
- NITTA Project founder

ryukzak.github.io

Agenda

Practices for continues quality control in NITTA project

I. Quality in Software System — question overview

II. The ASIP CAD Example — NITTA project overview

III. Development Process — development process overview

IV. Verification Methods — review of applied non widespread practices

Quality in Software System

Quality Concept in Software System

- Traditional engineering
  - “Quality is a conformance to requirements” — Philip Crossby
  - The system of quality is prevention
  - The performance standard is zero defects (relative to requirements)
  - The measurement of quality is the price of nonconformance
- In software engineering
  - “Quality is a value to some person” — Gerald Weinberg
  - Requirement engineering is a development process part
  - Tradeoffs between different stakeholders

Elements of Quality Software

- Product vision (understanding stakeholders and their needs, requirements)
- Project management (understanding priorities, processes, reaction to unexpected situation)
- **Routine development processes** (bureaucracy, regulation, automatization, continuous quality control, automatization)
- Acceptance tests (end-user testing to bring system onto utilization stage)

Cost, Time, and Quality

Tradeoff

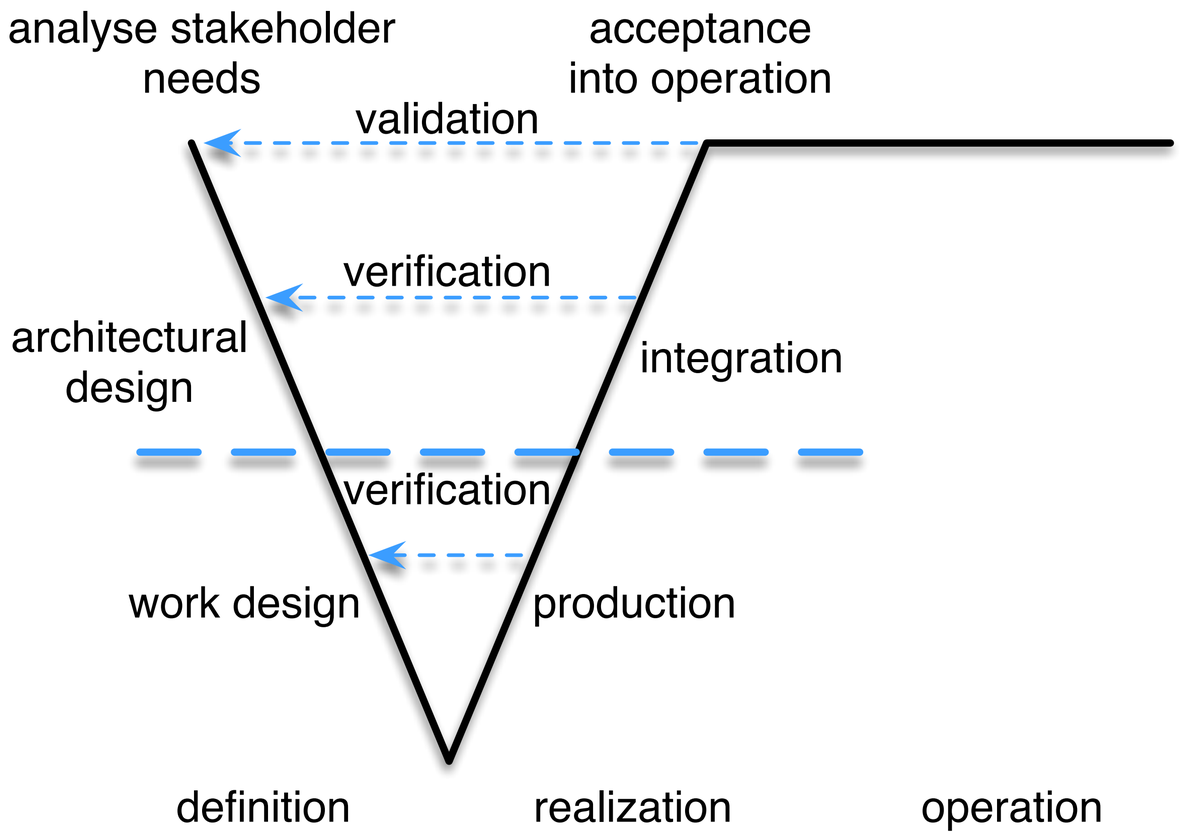
- Business needs (time, budget) are immediate shows stopper.
- Bad quality with bad management is not a show stopper due to the end of the project.
- Can we make absolute quality if we have infinite time and budget?

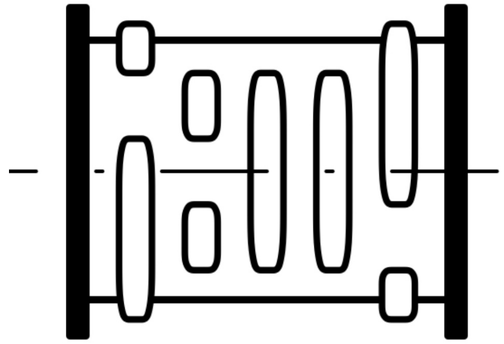
https://medium.com/@vivekmadurai/quality-time-and-money-39278f990092

Quality Control

Key processes

- Validation — checking system accordance to stakeholders' needs
- **Verification** — checking the system for compliance with a formalized requirements
  - Static verification (without execution, general properties)
  - Dynamic verification (with execution, specific data)





The ASIP CAD Example NITTA Project

ryukzak.github.io/projects/nitta

NITTA Project

As a Research Pet Project

- It is the ongoing project
- It will be published on Github in the middle of 2021 under the BSD license
- Pet means:
  - Just for Fun, the commercial outcome is not a priority at present
  - Abilities to ignore many commercial project restrictions
  - Non-regular contributions
  - Not deadline
- Research and University means:
  - Main goals: articles, conferences, bachelor/master/Ph.D. students
  - Abilities pay extra attention to a narrow question
  - Open requirement lists (at present)
  - The team mostly consist of students:
    - Not a professional, require mentoring and reviewing
    - Require fast feedback loop

NITTA Project

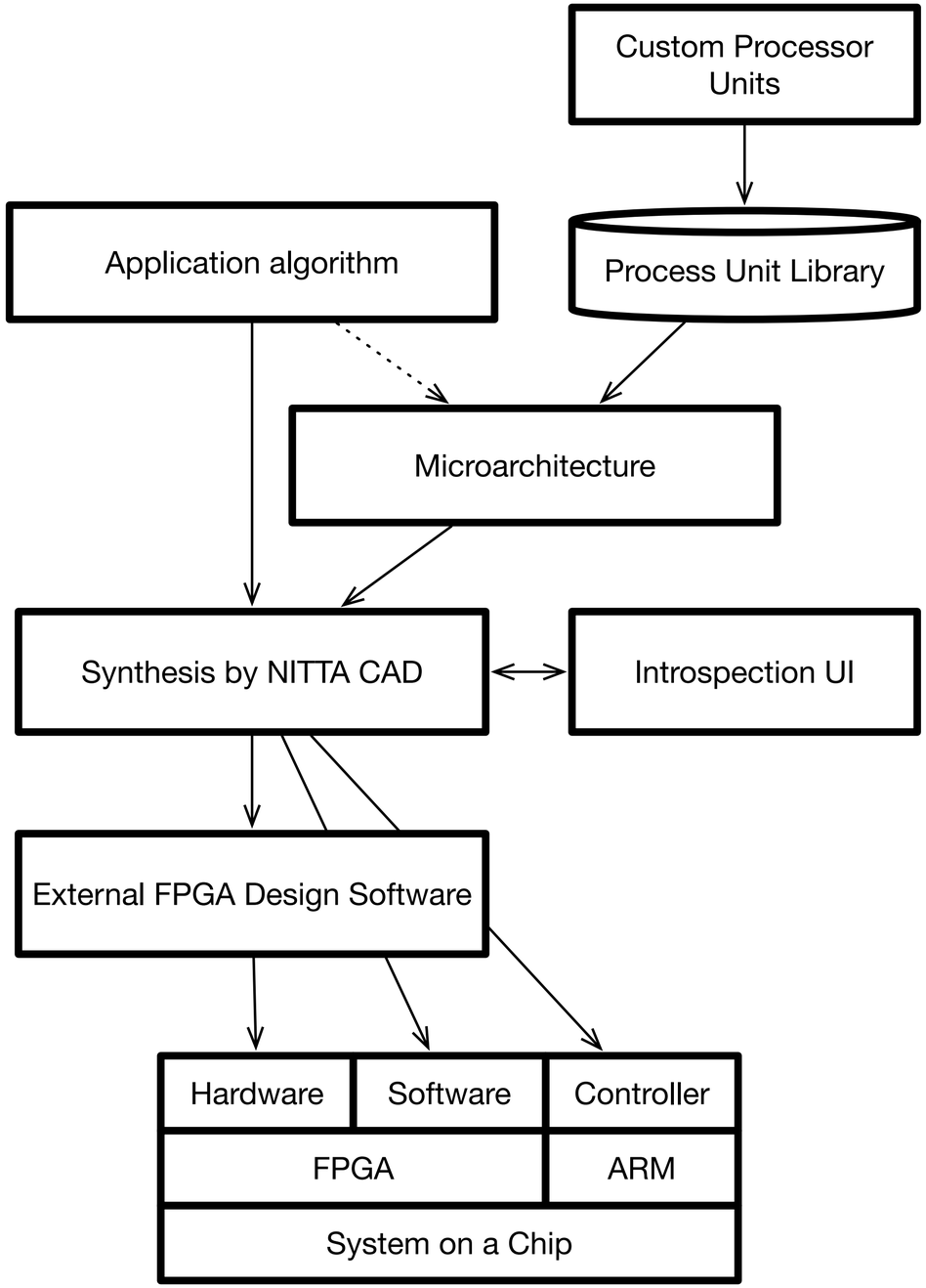
As a Product

- NITTA project is dedicated to developing the CAD for generating and programming hard real-time Application-Specific Processors with Coarse-Grained Reconfigurable Array Architecture for cyclic execution of control or signal/data processing algorithms. Application:
  - Development of embedded and cyber-physical systems
  - Hardware and software testing and rapid prototyping (HIL, PIL)
  - Development of accelerators and coprocessors (e.g., for System Dynamic)
- These processors are based on the original Not Instruction Transport Triggered Architecture (NITTA).
  - It provides high speed and parallel execution of irregular algorithms (where GPU is not applicable).
  - It makes reconfigurable processors for different application domains.
  - It provides a high-level language for application developers and fast compilation (Lua, *XMILE*).

NITTA Project

User Work Flow

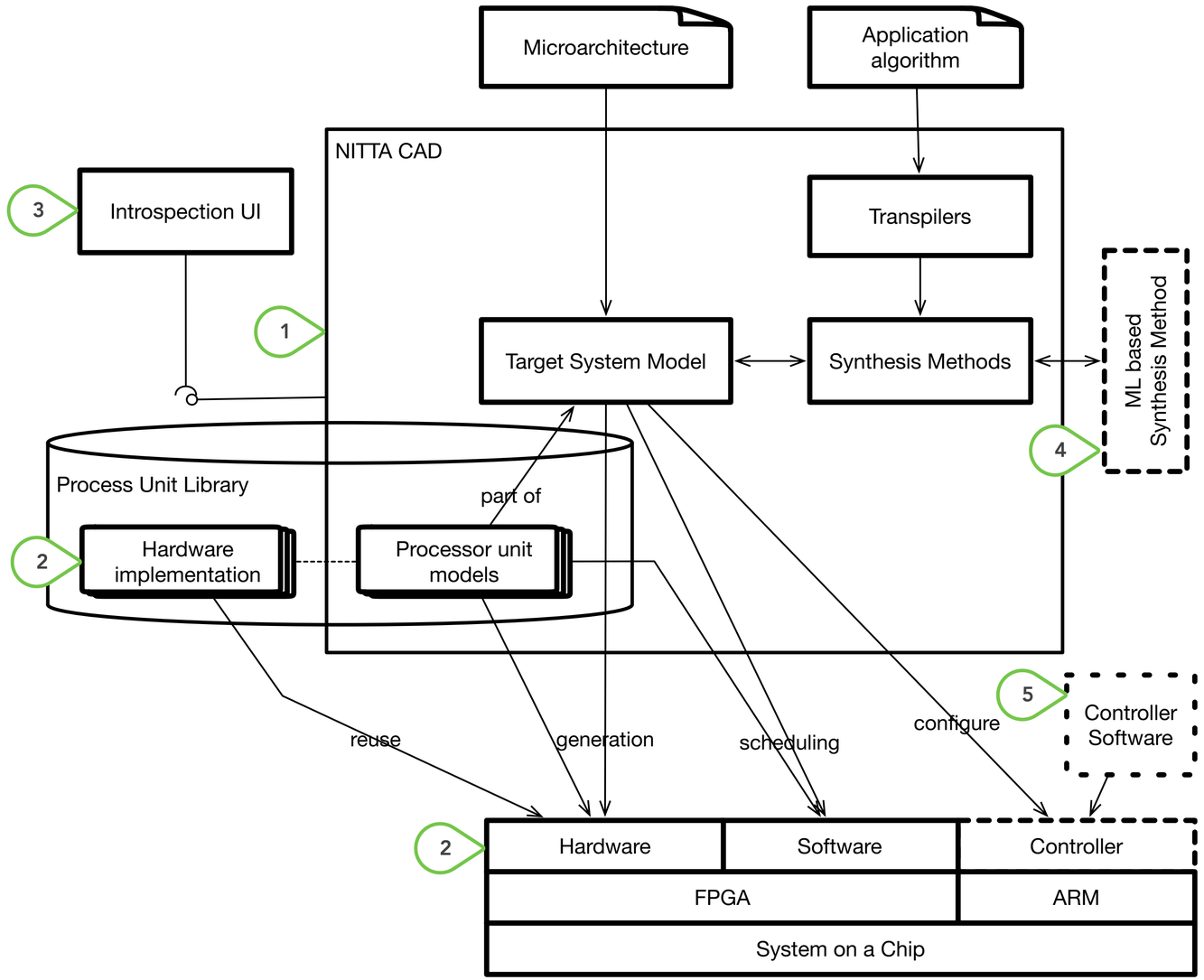
- Application algorithm — algorithm on high-level programming language
- Process Unit — element of processor, which performs data processing, storing, and IO.
- Microarchitecture — composition of process units, buses, interconnect
- Introspection UI — user interface for analysis and control over synthesis process.
- External FPGA Design Software — tool synthesis FPGA configuration from hardware description language (Quartus right now)
- FPGA (field-programmable gate array) — cheap custom hardware (board + preparing time)



NITTA Project

Development Stack

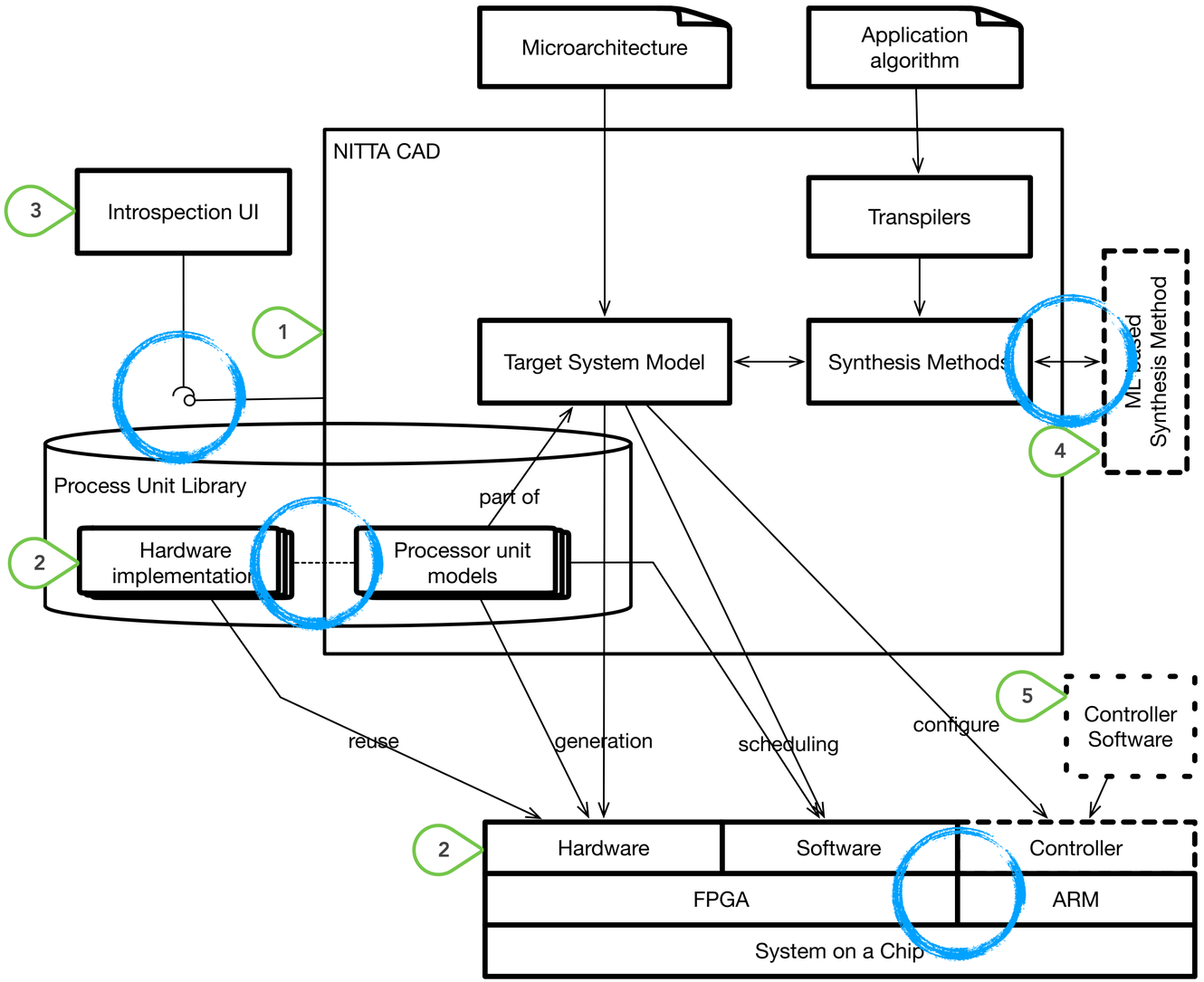
1. Haskell — CAD itself
2. FPGA, Verilog — hardware
3. Typescript + React — UI
4. Python — ML based Synthesis
5. Rust — control software (in future)



NITTA Project

Key Difficulties

- The extreme learning curve of the subject
- The hard learning curve for tools
- Strongly linked software components with continues changing specification
- Gaps between different technologies (User interface — CAD, Hardware — Models):
  - Integration issues
  - Misunderstanding between different developers
  - A lot of boilerplate code
- Late integration



Development Process

Development Process

Practices

- Weekly meeting
  - Dynamic development process management
  - Preventing sticking
  - Experience exchange
- Code Review by GitHub by mentor and team member
- Continues Integration
  - Source code auto-format
  - “-Wall”
  - Lint-tools
  - Unit and integration tests
  - Automatic documentation generation

Verification methods

Test by Interactive Example

- Problems:
  - Project documentation
  - Keeping the documentation up to date
  - Context related documentation to reduce the learning curve
- Solution: a doctest like testing approach (heavy spread in Python community)

Test by Interactive Example

doctest as an alternative to unittest

- “The doctest module searches for pieces of text that look like interactive Python sessions, and then executes those sessions to verify that they work exactly as shown.”
- Represented in some other development tools, e.g., C++, Haskell, Elixir, Elm, Rust
- It can be simple implemented in all languages with REPL
- Usage of integrated with documentation tests force to make documentation up to date
- Restriction: simple lifecycle, a small amount of input/output data

```
def factorial(n):
    """Return the factorial of n, an exact integer >= 0.

    >>> [factorial(n) for n in range(6)]
    [1, 1, 2, 6, 24, 120]
    >>> factorial(30)
    265252859812191058636308480000000
    >>> factorial(-1)
    Traceback (most recent call last):
        ...
    ValueError: n must be >= 0
      """

    import math
    if not n >= 0:
        raise ValueError("n must be >= 0")
    if math.floor(n) != n:
        raise ValueError("n must be exact integer")
    if n+1 == n:  # catch a value like 1e300
        raise OverflowError("n too large")
    result = 1
    factor = 2
    while factor <= n:
        result *= factor
        factor += 1
    return result
```

Python, Development Tools Documentation

End-to-End Static Typing

Problem statement

- Static typing is one of the best static invariant checkers for software with the appropriate cost.
- The initial choice of the development tool (Haskell) is justified by a powerful type system, which significantly simplifies control over project consistency.
- A heterogeneous system architecture (most of the complex software system) have gapped between different technologies.
- How to establish a typed interface between two statically typed components: CAD (Haskell) and UI (Typescript)?

End-to-End Static Typing

Available options:

- Manual implementation in accordance with API specification:
  - Require a lot of documentation work
  - Require tests for API verification with high coverage
  - Any change requires work on both side
  - Full control and less artificial restrictions on both side
  - A lot of boilerplate code
- Use static-typed language-neutral mechanism for serializing structured data, e.g., protocol buffer, ASN.1
  - Require formal specification of a transferred object and third party software for code generation or marshaling
  - Both sides were restricted by the serializing mechanism on conceptual and implementation level
  - A gap between transport and application levels

End-to-End Static Typing

Solution: server-driven code generation

- Applicable only in case if one component is derived from another:
  - Client-driven — generation server-side software on access patterns (see: backend as a service)
    - Preferable for developing mobile application with simple data storage
  - Server-driven — generation access library based on exposed API (our case, UI is derived from CAD).
  - It is preferable due to the possibility of multiple clients
  - Automatic API documentation generation
- Consistency check on the type-level
- Heavily restricted by used tools

- Our solution:
  - Third-party libraries: servant, servant-server, servant-js, servant-docs, aeson, aeson-typescript
  - Flow:
    - Native Haskell data types (part of the CAD)
    - Utility Haskell data types for infinite and redundant data types (manual)
    - JSON serialization (auto)
    - A set of generic typescript types (auto)
    - Marshaling between Haskell and HTTP API (auto)
    - Server API for JS on Axios (auto)
    - Mapping server API from JS to TypeScript (manual)

Domain Specific Language for Tests

Problem Statement

- Writing tests for units with complex input data, output data, and life-cycle requires a lot of boilerplate code
- That requires a lot of time for writing, reading, and maintaining tests
- A huge gap between application domain and technical implementation details are presented
- Tests tend to be not observable. Programmers can extract some parts of essential data from the test
- Tests tend to be not traceable and debug-able

Domain-Specific Language for Test

Solution: Application Level Domain-Specific Language

- Behavior-Driven Development (BDD), focuses:
  - Where to start in the process
  - What to test and what not to test
  - How much to test in one go
  - What to call the tests
  - How to understand why a test fails
- [embedded] Domain-Specific Language
  - More application-specific solution
  - The simpler learning curve in comparison with BDD, but not portable
- Allow writing tests with partial code reuse options.

```
Feature: Eating too many cucumbers may not be good for you

  Eating too much of anything may not be good for you.

  Scenario: Eating a few is no problem
    Given Alice is hungry
    When she eats 3 cucumbers
    Then she will be full
```

https://cucumber.io

```
puUnitTestCase "multiplier test" pu $ do  -- 1. Created test case for provided PU
    assign $ multiply "a" "b" ["c", "d"]  -- 2. Bind function 'a * b = c = d' to PU
    setValue "a" 2                        --    Set initial input values
    setValue "b" 7                        --    for further CoSimulation

    decideAt 1 2 $ consume "a"            -- 3. Bind input variable "a" from 1 to 2 tick
    decide $ consume "b"                  --    Bind input variable "b" at nearest tick
    decideAt 5 5 $ provide ["c"]          --    Bind output variable "c" at 5 tick
    decide $ provide ["d"]                --    Bind output variable "d" at nearest tick

    traceProcess                          --    Print current process state to console

    assertSynthesisDone                   -- 4. Check that all decisions are made
    assertCoSimulation                    --    Run CoSimulation for current PU
```

from NITTA Project

Property-Based Testing and CoSimulation

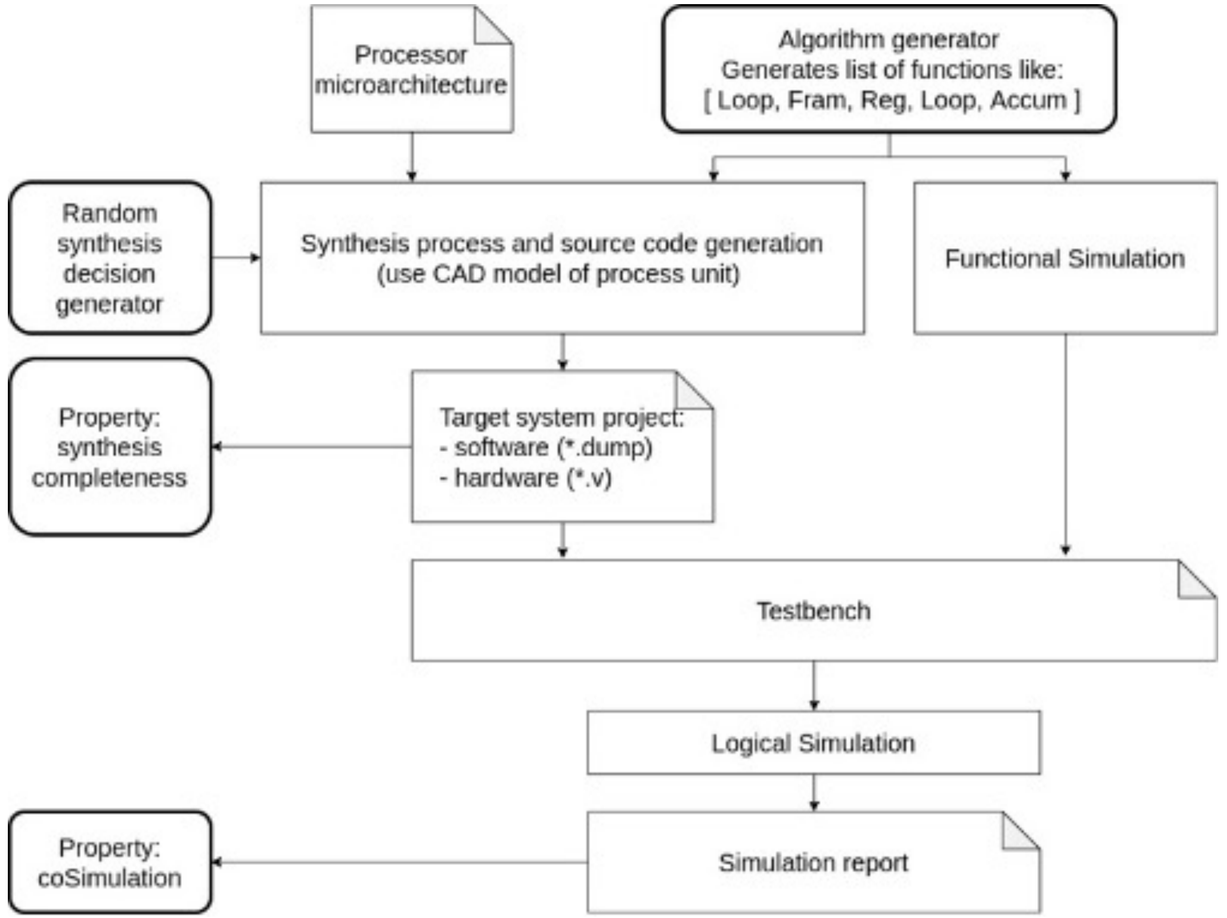
Problem Statement

- Development tools have a very complex and heavy variety of input data
- Any processor unit is two machines (hardware implementation and CAD model) that should be consistent to each other with:
  - Multiple supported functions
  - Own instruction set
  - Possible concurrent function execution
  - Possible internal resources
- Late integration: to check the correctness of CAD, process unit hardware implementation, and its model, we need to produce and run the target system.
- How to prepare enough amount of test cases?

Property-Based Testing

Options and Solution

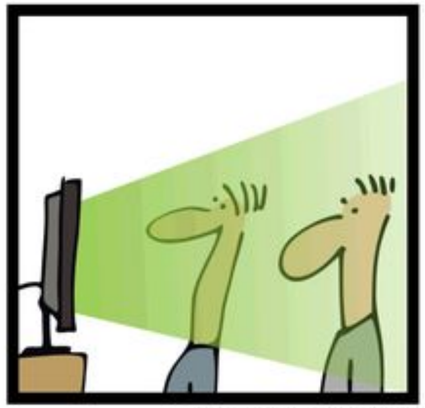
- Certified programming as a static way to check general system' properties
- Property-Based Testing (PBT) as a dynamic verification method.
- The main idea: if we can not prove properties for a general case, we can do it for a large amount of autogenerated data.
- Key task: define general unit properties. E.g.,
  - `list = reverse(reverse(list))`
  - `(a + b) + c = a + (b + c)`
  - All algorithm' function should be scheduled for execution.
  - Results of functional and logical simulation should equal



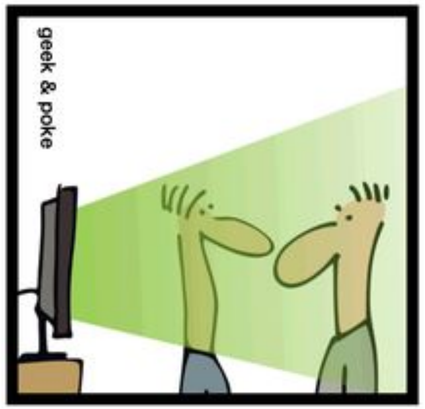
More details in the article:
Verification of the CAD System for an Application-Specific Processor by Property-Based Testing

Tests for Tests

- Complex test utilities can contain errors themselves.
- Worst case: tests passed, but tests are not actually check anything and create a false sense of trust.
- Without continuously checking, we can miss the moment when test utilities have been broken.
- Solution:
  - Embedded special components into the project to imitate common error types and catch them by routine automated tests.



HAVING A GREEN HUDSON/JENKINS ...



... IS ...

... A GREAT EXPERIENCE

Conclusion

- Automatization can be used to replace and form a development culture.
- Merging documentation and unit tests in a literate style can improve both.
- End-to-end static typing across different technologies can be implemented by code generation. It reduces the amount of glue boilerplate code.
- Application of [embedded] Domain-Specific Languages can significantly reduce the complexity and NLoC of your tests.
- Property-Based Testing can significantly increase test coverage for a complex algorithm without writing many test cases if you can define even simple properties.
- It is not acceptable to trust your tests if they're not deadly simple.

Thank you!

ryukzak.github.io

Ph.D. Aleksandr Penskoi, ITMO University, Russia